Diabetes Analysis Prediction .

importing necessary library

In [200…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [201…
```
df = pd.read_csv("diabetes.csv")
```

In [202…
```
df
```

Out[202]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	
...	...	...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.171	63	
764	2	122	70	27	0	36.8	0.340	27	
765	5	121	72	23	112	26.2	0.245	30	
766	1	126	60	0	0	30.1	0.349	47	
767	1	93	70	31	0	30.4	0.315	23	

768 rows × 9 columns

Data details : 1) Pregnancies: NUmber of times pergnant 2) Glucose: Refers to the blood sugar level measure 3) BloodPressure: Diastolic blood pressure(mm Hg) 4) SKinthinkness: Triceps skin fold thickness(mm) 5) Insulin: serum insulin (mu U/ml) 6) BMI: Body mass index(weight in kg/Height in m ^2) 7) Diabetespedigreefunction:unction which scores likelihood of diabetes based on family history 8) Age: In years 9) Outcomes: Class variable (0-non-diabetic,1-diabetic)

In [203…
```
df.shape
```

Out[203]:
```
(768, 9)
```

In [204…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Checking Null Values.

In [205…

```
df.isnull().sum()
```

Out[205]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [206…

```
df.describe()
```

Out[206]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.4718
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.3313
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0780
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.2437
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.3725
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.6262
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.4200

In [207…

```
#df=df.iloc[1:10,]
```

Visualization.

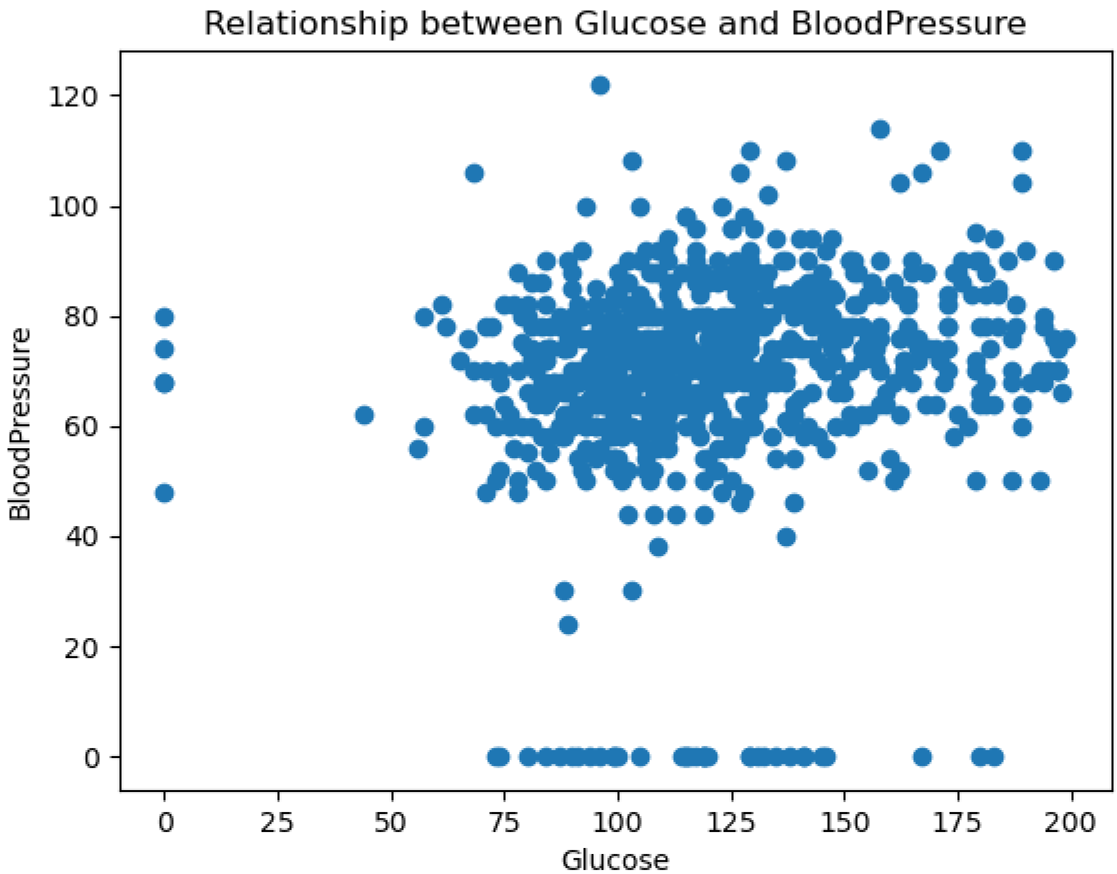
In [208…

```
plt.scatter(df['Glucose'],df['BloodPressure'])
plt.xlabel("Glucose")
plt.ylabel("BloodPressure")
plt.title("Relationship between Glucose and BloodPressure")
```

Out[208]:

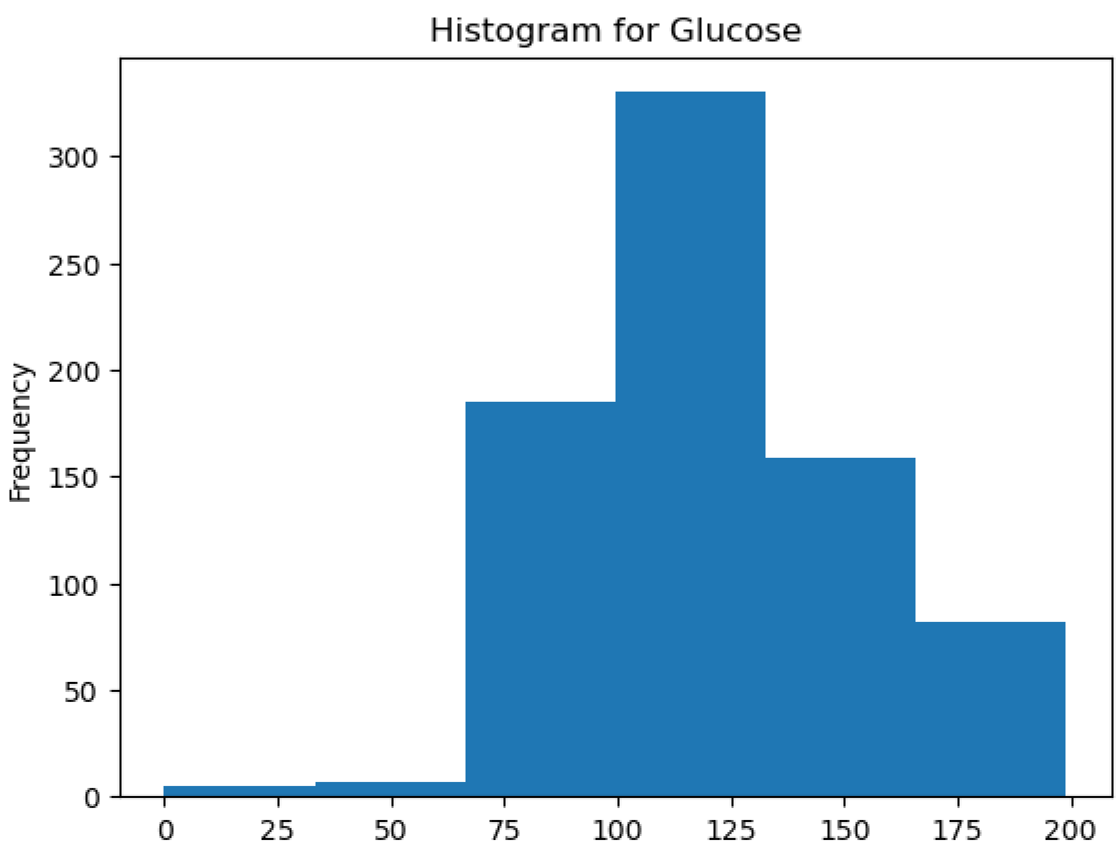
```
Text(0.5, 1.0, 'Relationship between Glucose and BloodPressure')
```

Loading [MathJax]/extensions/Safe.js



In [209…

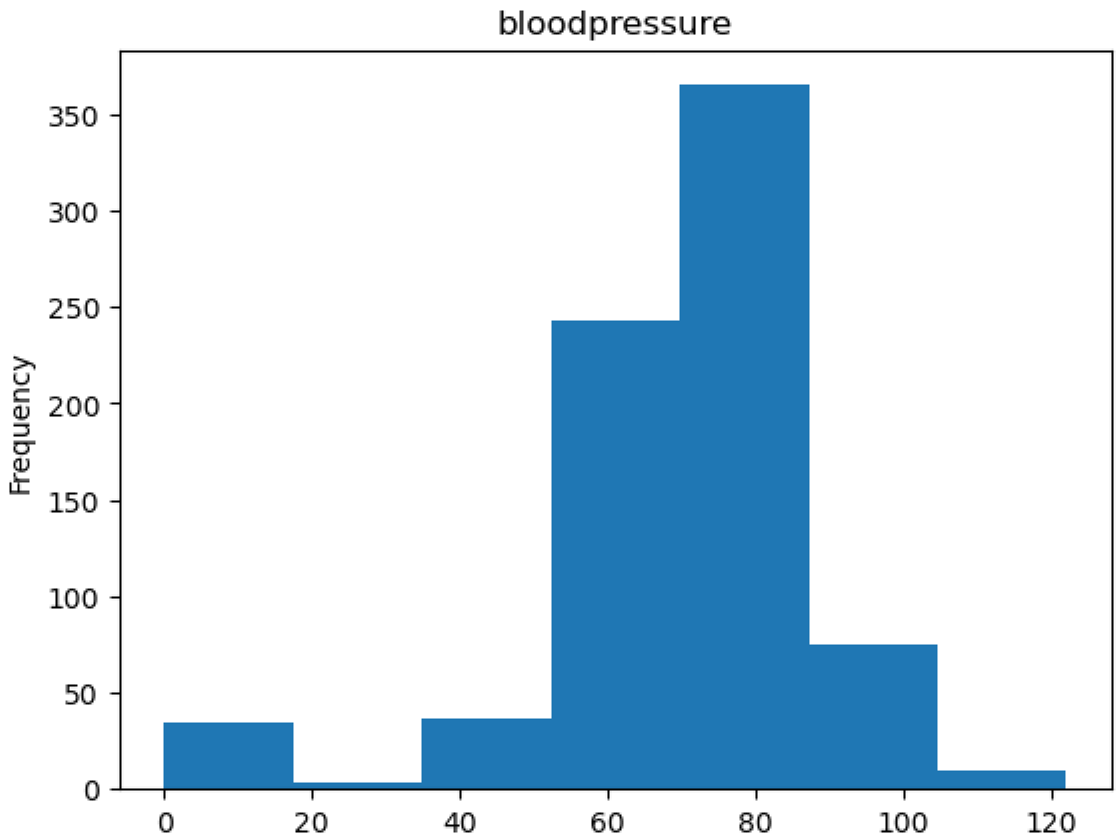
```
df['Glucose'].plot(kind='hist',bins=6)
plt.title("Histogram for "+df['Glucose'].name)
plt.show()
```



In [210…

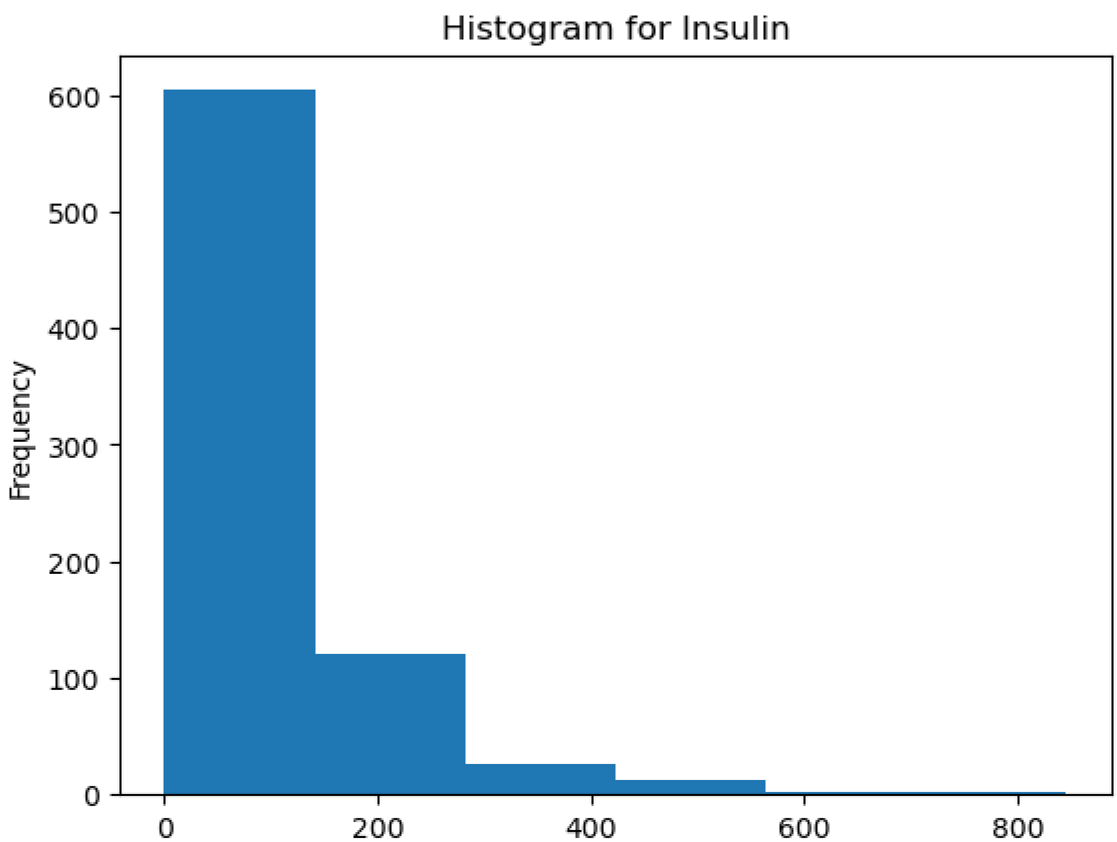
```
df['BloodPressure'].plot(kind='hist',bins=7)
plt.title("bloodpressure")
plt.show()
```

Loading [MathJax]/extensions/Safe.js



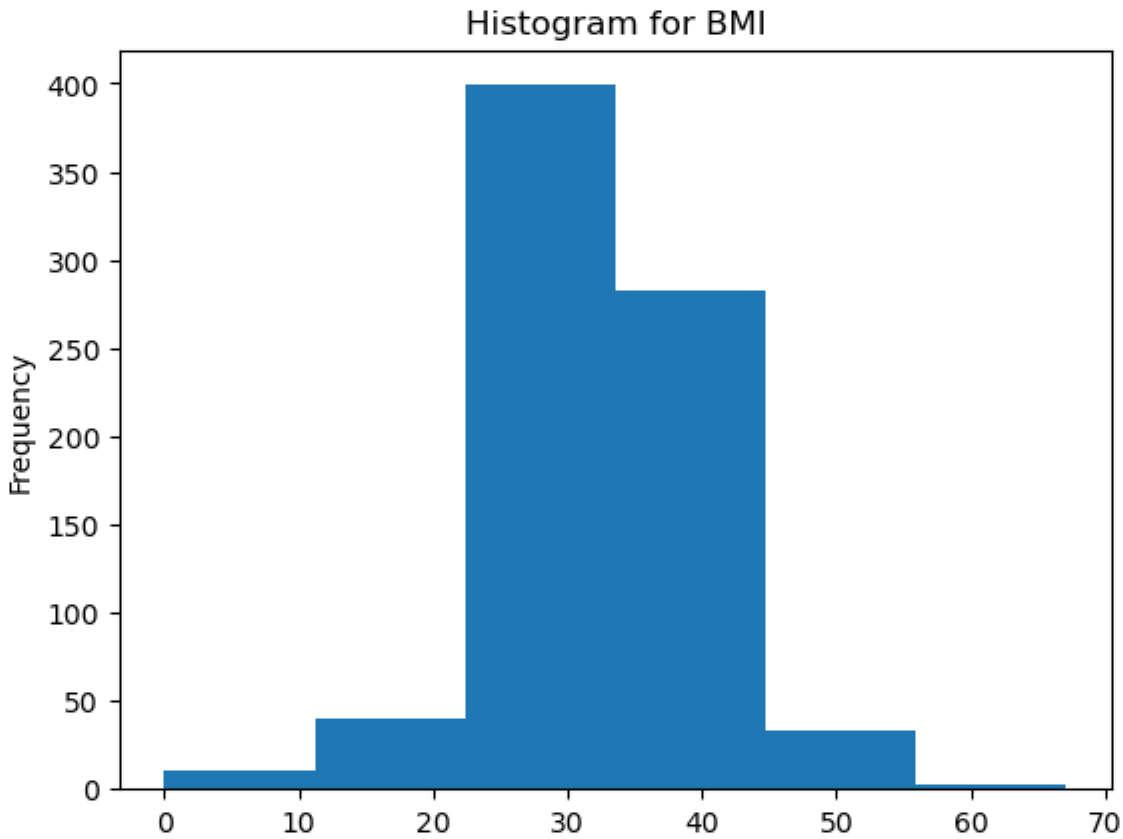
In [211…

```
df['Insulin'].plot(kind='hist',bins=6)
plt.title("Histogram for "+df['Insulin'].name)
plt.show()
```



In [212…

```
df['BMI'].plot(kind='hist',bins=6)
plt.title("Histogram for "+df['BMI'].name)
plt.show()
```



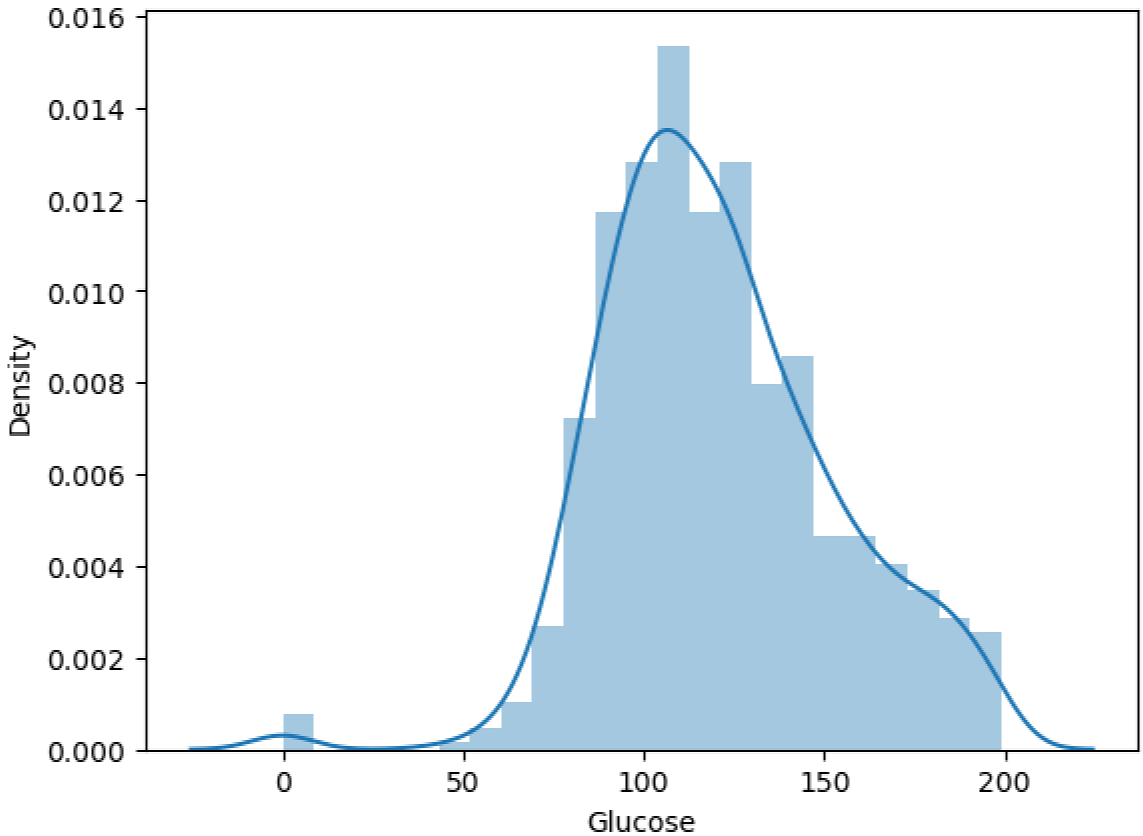
In [213…
```
#df = df.iloc[1:1000]
```

In [214…
```
sns.distplot(df['Glucose'])
```

```
C:\Users\NANDHINI\AppData\Local\Temp\ipykernel_5912\2512680699.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Glucose'])
```

Out[214]:
```
<Axes: xlabel='Glucose', ylabel='Density'>
```

Loading [MathJax]/extensions/Safe.js

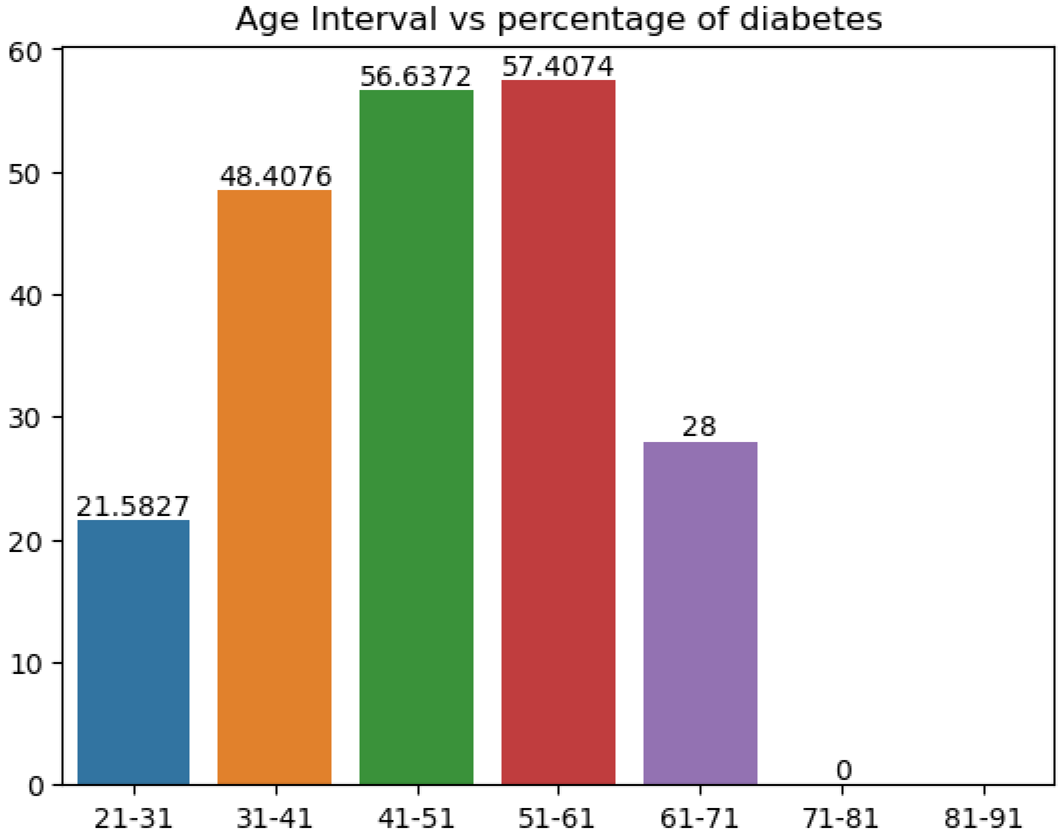


In [215…
```
s=df.groupby('Age')['Outcome'].sum()
s1=df['Age'].value_counts().sort_index()
d={f'{i+21}-{i+31}':(s[i:i+10].sum()/s1[i:i+10].sum()) for i in range(0,61,10)}
```

```
C:\Users\NANDHINI\AppData\Local\Temp\ipykernel_5912\2881464474.py:3: RuntimeWarning: inv
alid value encountered in scalar divide
  d={f'{i+21}-{i+31}':(s[i:i+10].sum()/s1[i:i+10].sum()) for i in range(0,61,10)}
```

In [216…
```
s=pd.Series(d)
ax=sns.barplot(x=s.index,y=s.values*100)
for i in ax.containers:
    ax.bar_label(i,)
plt.title('Age Interval vs percentage of diabetes')
```

Out[216]:
```
Text(0.5, 1.0, 'Age Interval vs percentage of diabetes')
```

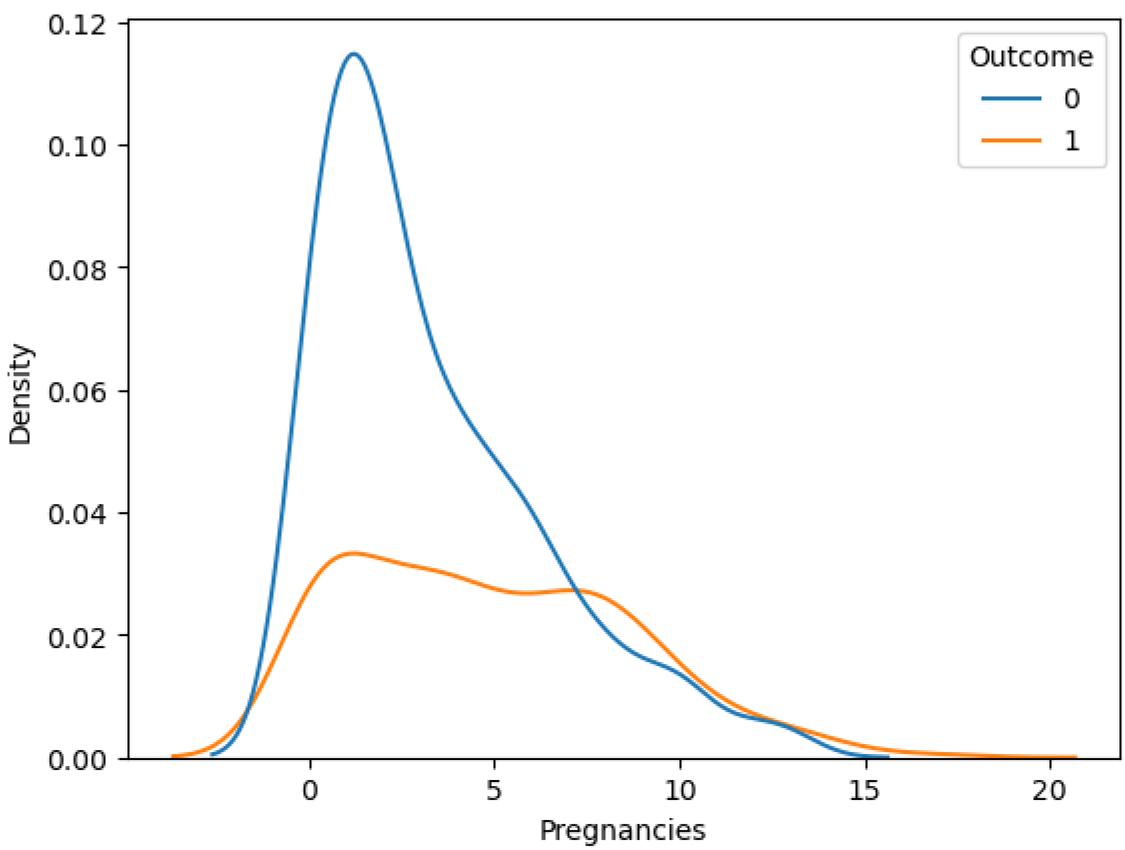


In [217…

```
sns.kdeplot(x='Pregnancies',hue='Outcome',data=df)
```

Out[217]:

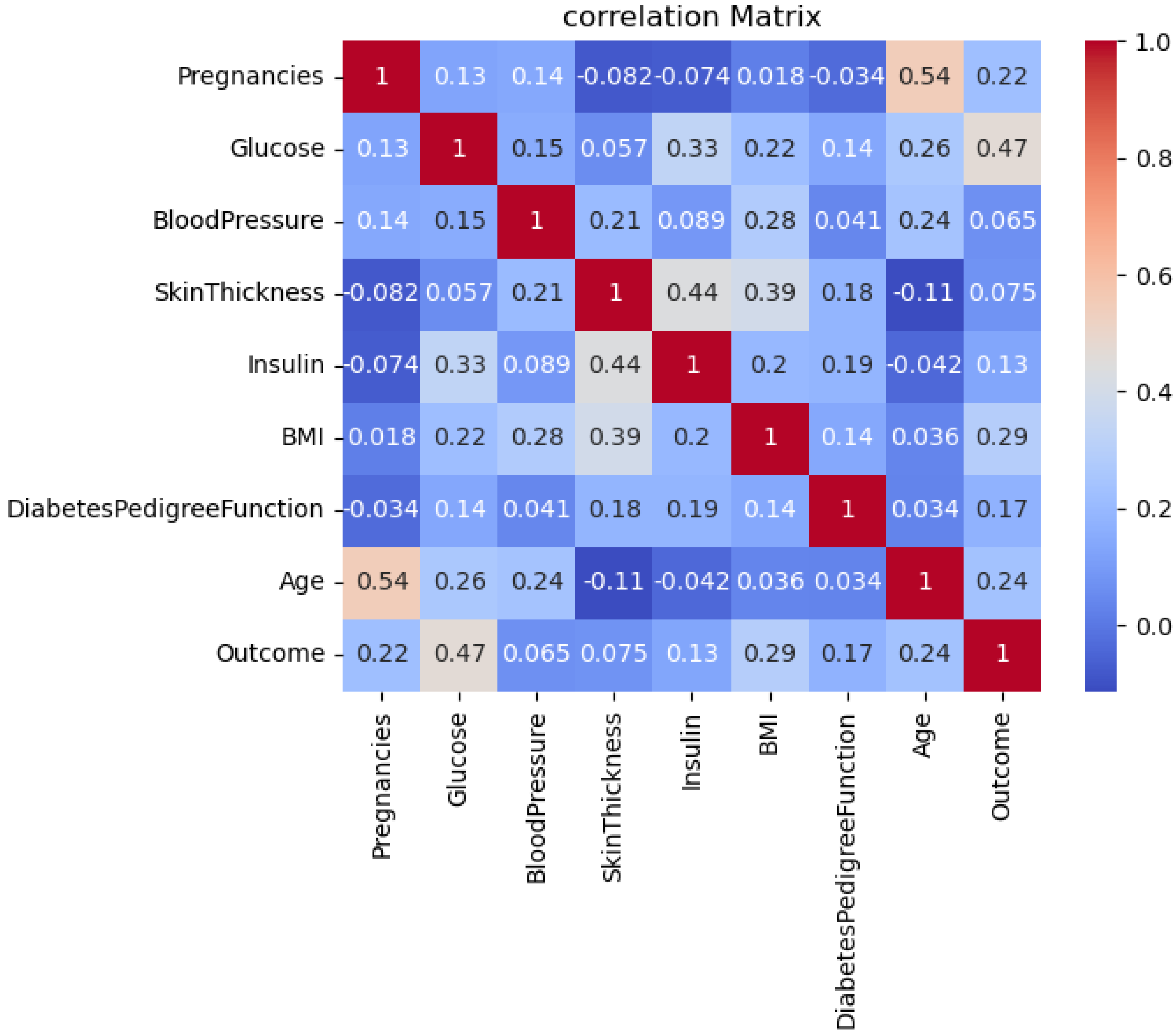
```
<Axes: xlabel='Pregnancies', ylabel='Density'>
```



In [218…

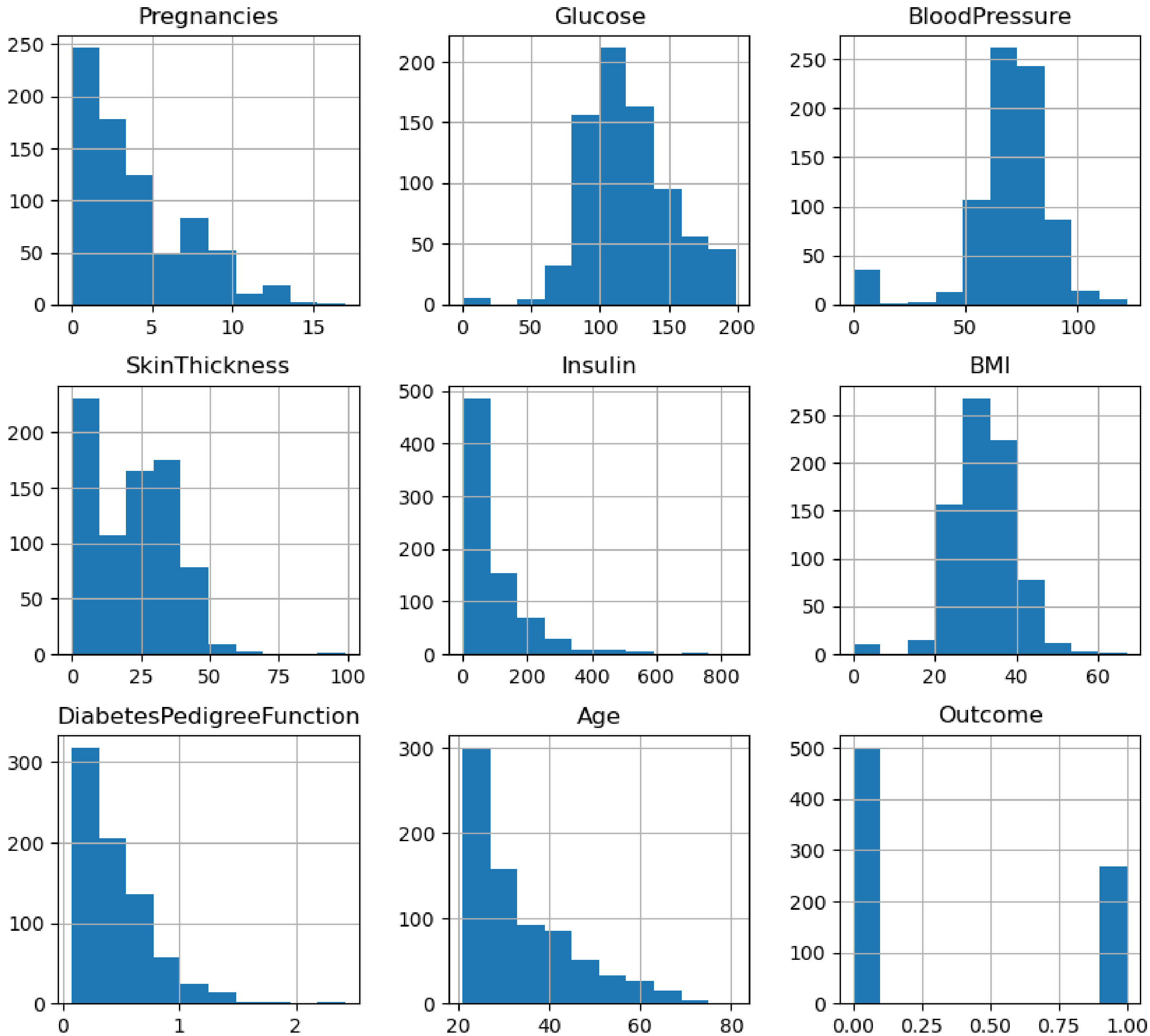
```
sns.heatmap(df.corr(), annot=True ,cmap = 'coolwarm')
plt.title('correlation Matrix')
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [219…

```
#creating histograms after replacing the zeros
df.hist(figsize=(10,9))
plt.show()
```



In [220…
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [221…
```
x = df.iloc[:, :8]
x = StandardScaler().fit_transform(x)
x
```

Out[221]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

In [222…
```
y = df.iloc[:, -1].values
y
```

Out[222]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

In [223…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state
```

In [224…
```
def perform(y_pred):
    print("Precision : ", precision_score(y_test, y_pred, average = 'micro'))
    print("Recall : ", recall_score(y_test, y_pred, average = 'micro'))
    print("Accuracy : ", accuracy_score(y_test, y_pred))
    print("F1 Score : ", f1_score(y_test, y_pred, average = 'micro'))
    cm = confusion_matrix(y_test, y_pred)
    print("\n", cm)
    print("\n")
    print("**"*27 + "\n" + " "* 16 + "Classification Report\n" + "**"*27)
    print(classification_report(y_test, y_pred))
    print("**"*27+"\n")

    cm = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = ['Non-Diabetic',
    cm.plot()
```

Gaussian Navie Bayes.

In [225…
```
from sklearn.naive_bayes import GaussianNB, BernoulliNB
```

In [226…
```
model_gnb = GaussianNB()
model_gnb.fit(x_train, y_train)
```

Out[226]: ▼ GaussianNB

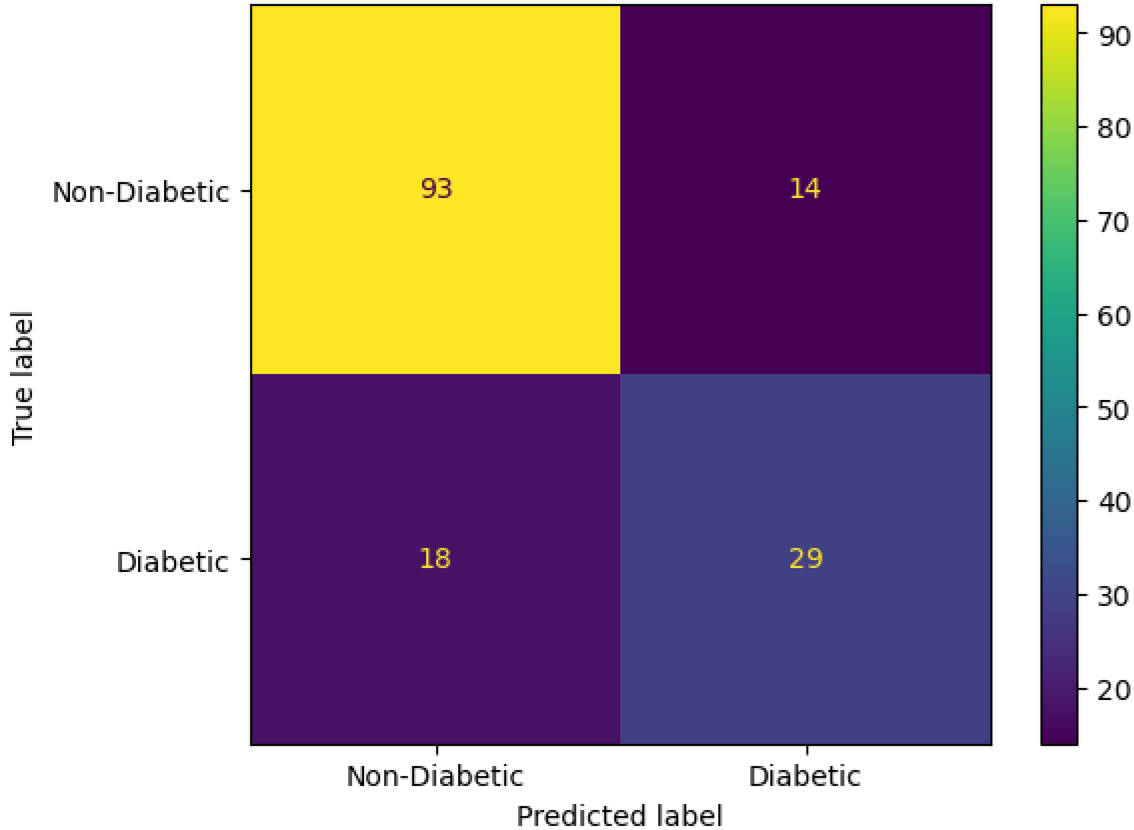
GaussianNB()

In [227…
```
y_pred_gnb = model_gnb.predict(x_test)
```

In [230…
```
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, ExtraTreesClass
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, NuSVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, cla
```

In [231…
```
perform(y_pred_gnb)
```

```
Precision :  0.7922077922077922
Recall :  0.7922077922077922
Accuracy :  0.7922077922077922
F1 Score :  0.7922077922077922

 [[93 14]
 [18 29]]


******************************************************
                Classification Report
******************************************************
              precision    recall  f1-score   support

           0       0.84      0.87      0.85       107
           1       0.67      0.62      0.64        47

    accuracy                           0.79       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.79      0.79      0.79       154

******************************************************
```



Bernoulii Navie Bayes.

In [233…
```
model_bnb = BernoulliNB()
model_bnb.fit(x_train, y_train)
```

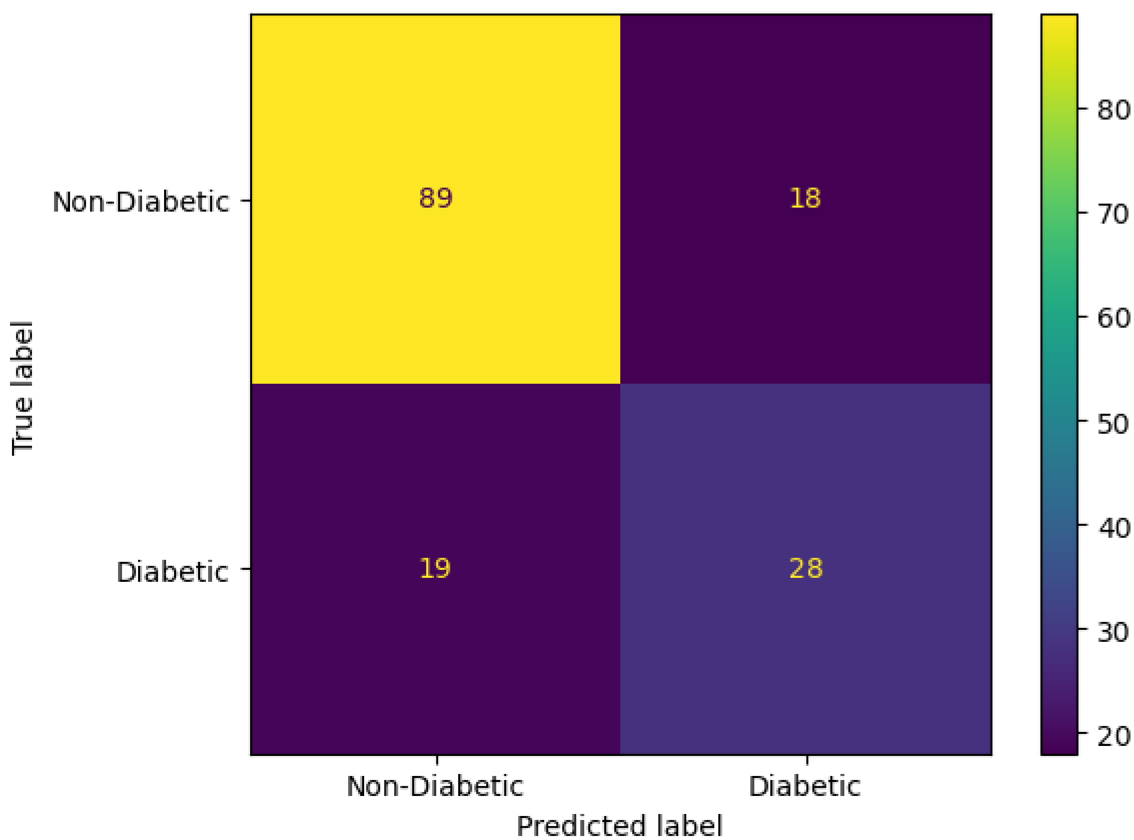
Out[233]:
```
▼ BernoulliNB
BernoulliNB()
```

In [234…
```
y_pred_bnb = model_bnb.predict(x_test)
```

In [235…
```
perform(y_pred_bnb)
```

Loading [MathJax]/extensions/Safe.js

```
Precision :  0.7597402597402597
Recall :  0.7597402597402597
Accuracy :  0.7597402597402597
F1 Score :  0.7597402597402597

 [[89 18]
 [19 28]]


*******************************************************
                Classification Report
*******************************************************
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       107
           1       0.61      0.60      0.60        47

    accuracy                           0.76       154
   macro avg       0.72      0.71      0.72       154
weighted avg       0.76      0.76      0.76       154

*******************************************************
```



Logistic Regression

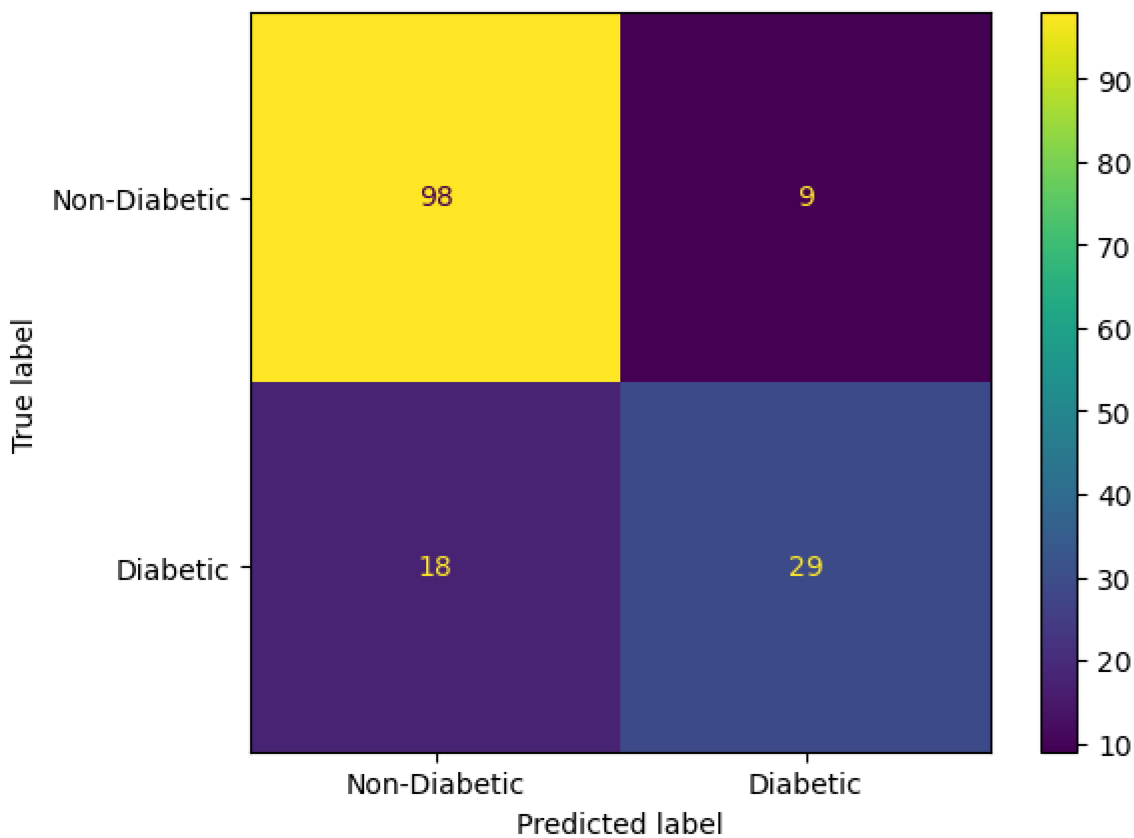
In [236…
```
model_lr = LogisticRegression()
model_lr.fit(x_train, y_train)
```

Out[236]:
```
▼ LogisticRegression
LogisticRegression()
```

In [237…
```
y_pred_lr = model_lr.predict(x_test)
```

In [238…
```
perform(y_pred_lr)
```

```
Precision :  0.8246753246753247
Recall :  0.8246753246753247
Accuracy :  0.8246753246753247
F1 Score :  0.8246753246753247

 [[98  9]
 [18 29]]


******************************************************
                Classification Report
******************************************************
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.62      0.68        47

    accuracy                           0.82       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.82      0.82      0.82       154

******************************************************
```



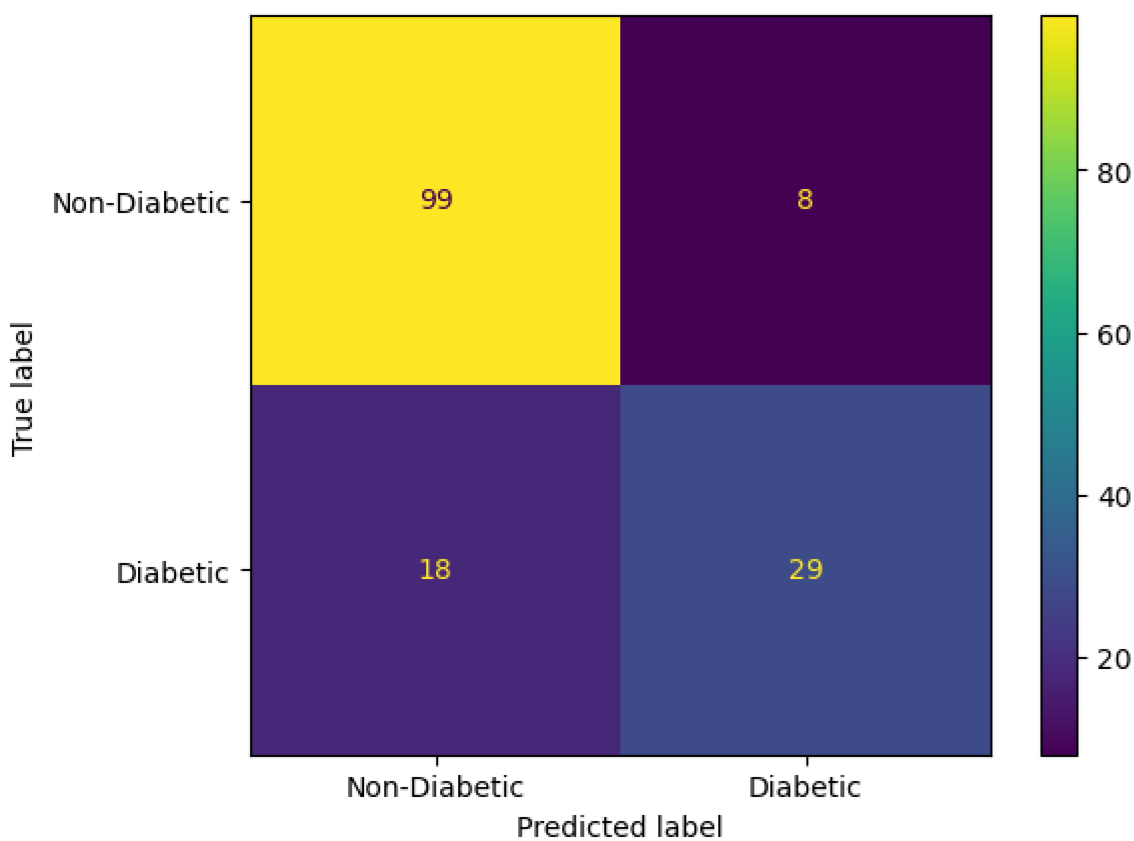
In [241…
```
model_rdg = RidgeClassifier()
model_rdg.fit(x_train, y_train)
```

Out[241]:
```
▼ RidgeClassifier
RidgeClassifier()
```

In [242…
```
y_pred_rdg = model_rdg.predict(x_test)
```

In [243…
```
perform(y_pred_rdg)
```

```
Precision :  0.8311688311688312
Recall :  0.8311688311688312
Accuracy :  0.8311688311688312
F1 Score :  0.8311688311688312

 [[99  8]
 [18 29]]


******************************************************
                Classification Report
******************************************************
              precision    recall  f1-score   support

           0       0.85      0.93      0.88       107
           1       0.78      0.62      0.69        47

    accuracy                           0.83       154
   macro avg       0.81      0.77      0.79       154
weighted avg       0.83      0.83      0.82       154

******************************************************
```



In [244…
```
model_dt = DecisionTreeClassifier()
model_dt.fit(x_train, y_train)
```

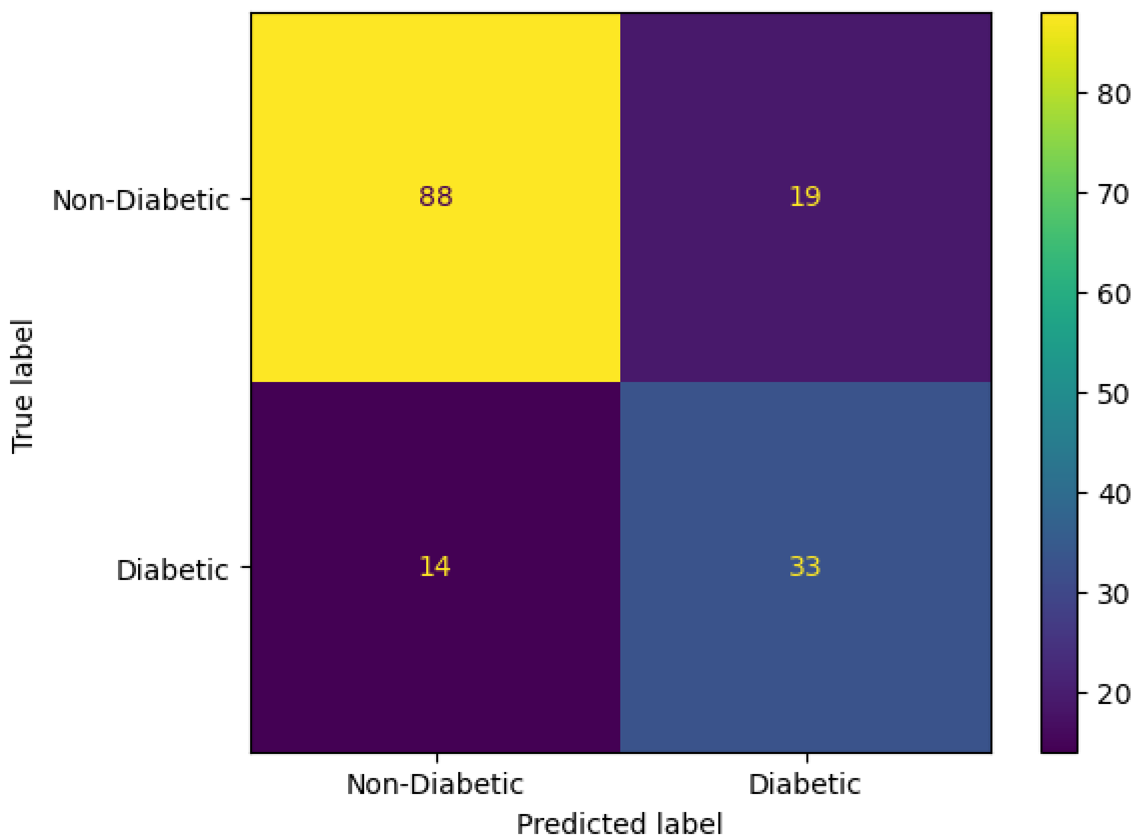
Out[244]:
```
▼ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [245…
```
y_pred_dt = model_dt.predict(x_test)
```

Loading [MathJax]/extensions/Safe.js

In [246…
```
perform(y_pred_dt)
```

```
Precision :  0.7857142857142857
Recall :  0.7857142857142857
Accuracy :  0.7857142857142857
F1 Score :  0.7857142857142857

 [[88 19]
 [14 33]]


******************************************************
                Classification Report
******************************************************
              precision    recall  f1-score   support

           0       0.86      0.82      0.84       107
           1       0.63      0.70      0.67        47

    accuracy                           0.79       154
   macro avg       0.75      0.76      0.75       154
weighted avg       0.79      0.79      0.79       154

******************************************************
```



In [247…
```
model_rf = RandomForestClassifier()
model_rf.fit(x_train, y_train)
```

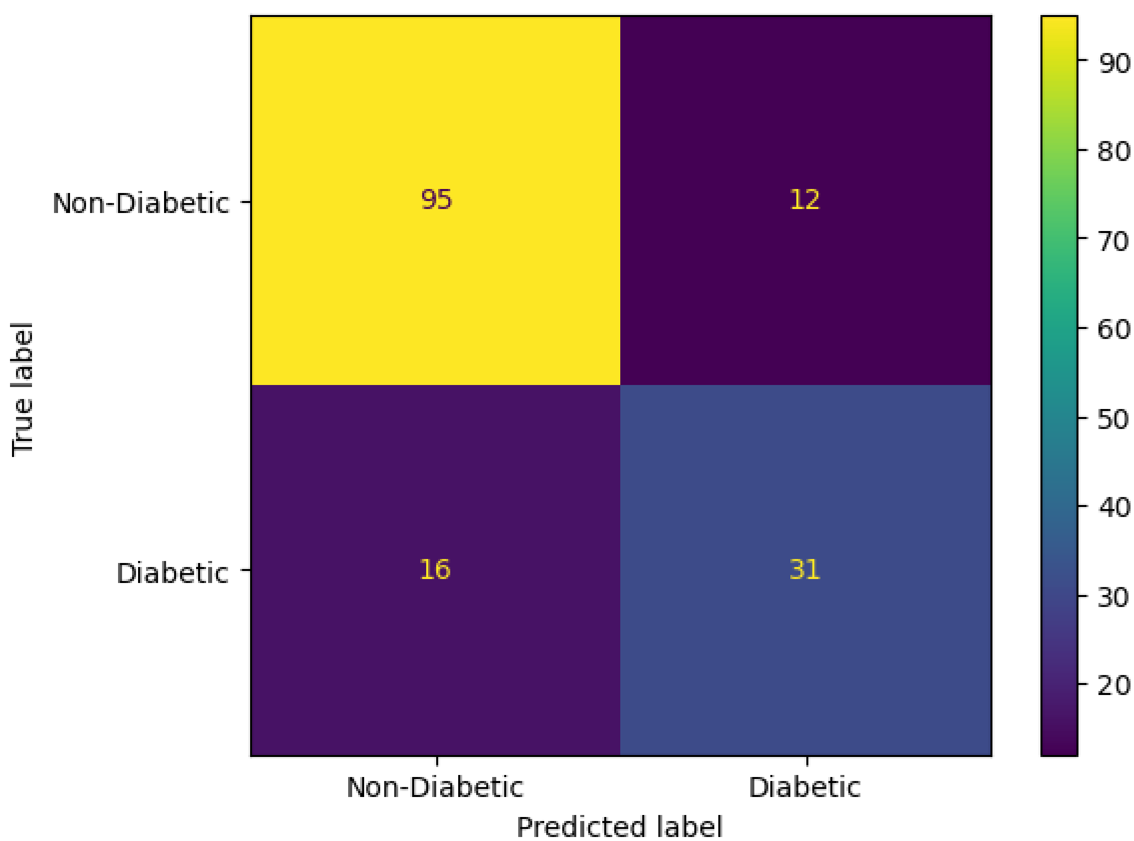
Out[247]:
```
▼ RandomForestClassifier
RandomForestClassifier()
```

In [248…
```
y_pred_rf = model_rf.predict(x_test)
```

Loading [MathJax]/extensions/Safe.js

In [249…
```
perform(y_pred_rf)
```

```
Precision :  0.8181818181818182
Recall :  0.8181818181818182
Accuracy :  0.8181818181818182
F1 Score :  0.8181818181818182

 [[95 12]
 [16 31]]


******************************************************
                Classification Report
******************************************************
              precision    recall  f1-score   support

           0       0.86      0.89      0.87       107
           1       0.72      0.66      0.69        47

    accuracy                           0.82       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.81      0.82      0.82       154

******************************************************
```



In [250…
```
model_ada = AdaBoostClassifier()
model_ada.fit(x_train, y_train)
```

Out[250]:
```
▼ AdaBoostClassifier
AdaBoostClassifier()
```

In [251…
```
y_pred_ada = model_ada.predict(x_test)
```

In [252…

```
perform(y_pred_ada)
```

```
Precision :  0.7792207792207793
Recall :  0.7792207792207793
Accuracy :  0.7792207792207793
F1 Score :  0.7792207792207793

 [[89 18]
 [16 31]]


******************************************************
                Classification Report
******************************************************
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       107
           1       0.63      0.66      0.65        47

    accuracy                           0.78       154
   macro avg       0.74      0.75      0.74       154
weighted avg       0.78      0.78      0.78       154

******************************************************
```